Computer Sciences Department

The Influence of Random Delays on Parallel Execution Times

Vikram S. Adve
Mary K. Vernon

Technical Report #1138

February 1993

UNIVERSITY OF
WISCONSIN
MADISON

The Influence of Random Delays on Parallel Execution Times†

Vikram S. Adve
Mary K. Vernon

Computer Sciences Technical Report #1138
February 1993

† To appear in the *Proceedings of the 1993 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems.*

The Influence of Random Delays on Parallel Execution Times[†,*]

Vikram S. Adve *and* Mary K. Vernon

University of Wisconsin-Madison
Computer Sciences Department
1210 West Dayton Street
Madison WI 53706.

{adve,vernon}@cs.wisc.edu

Abstract. Stochastic models are widely used for the performance evaluation of parallel programs and systems. The stochastic assumptions in such models are intended to represent non-deterministic processing requirements as well as random delays due to inter-process communication and resource contention. In this paper, we provide compelling analytical and experimental evidence that in current and foreseeable shared-memory programs, communication delays introduce negligible variance into the execution time between synchronization points. Furthermore, we show using direct measurements of variance that other sources of randomness, particularly non-deterministic computational requirements, also do not introduce significant variance in many programs. We then use two examples to demonstrate the implications of these results for parallel program performance prediction models, as well as for general stochastic models of parallel systems.

1. Introduction

The execution of a parallel program on a multiprocessor system is influenced by a number of non-deterministic factors. These can include *non-deterministic processing requirements* (if the CPU requirements of the program vary significantly across different executions on a particular input), as well as *random delays* due to inter-process communication events and contention for shared hardware and software resources. Performance models of parallel programs and systems have typically used stochastic task execution times to represent such non-determinism. In many parallel programs, however, the CPU requirements are fixed or almost fixed for any particular input. (We provide some evidence for this later in the paper.) For such programs, does the variability due to random delays justify the stochastic assumptions in these performance models? More generally, how do random delays influence the variance of execution time of synchronizing processes in parallel programs? The answers to these questions should be useful not only for performance evaluation, but for programmers as well. To our knowledge, however, these questions have not previously been addressed.

In this work, we first attempt to quantify the non-determinism introduced by random delays in shared-memory parallel programs. We describe a renewal model of program behavior that can be used to evaluate the variance and distribution of the execution time of a process between synchronization points, in the presence of random delays. The model yields a simple estimate for the variance in terms of basic and intuitive parameters. We apply the model to different phases of several shared-memory programs, using detailed measurements to obtain the necessary parameter values. We also present direct measurements of the variance and distribution of process execution times, which include variability due to processing requirements as well. We use these to compare the relative influence of the two sources of non-determinism, as well as to evaluate the overall variance and distribution of execution time.

[†] This work is supported in part by the National Science Foundation (CDA-9024618, CDA-8920777, CCR-9024144) and by an IBM Graduate Fellowship.

[*] This paper appears in the *Proceedings of the 1993 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems.*

One key conclusion we obtain is that communication delays *do not introduce significant variance into the execution time of a process between synchronization points.* This holds for all phases of all programs we have analyzed, even when resource contention is severe and causes relatively large delays during program execution. Furthermore, the overall variance (due to both random delays and variations in processing requirements) is also extremely small in all but one program that we measured.[1]

We explore in some detail the implications of these results for parallel program performance prediction models as well as for more general stochastic models of parallel systems. One implication is that it appears possible to use a *deterministic* model for parallel program performance prediction. We show the potential advantages of this approach by comparing a simple deterministic model based on the above observations to some previous stochastic models [9, 10, 15, 19, 25] for one parallel program. A number of stochastic models have also been used to study more general performance aspects of parallel systems [3, 5, 11-13, 17-19, 22, 28]. Many of these models assume exponentially distributed task execution times to permit tractable solutions. Our work implies that it is important to determine if a result of such a model is strongly dependent on the exponential task assumption; in particular, such a result will not be applicable to many parallel programs. We give one example each of previous results that are and are not strongly dependent on such an assumption. Finally, we briefly discuss an implication of our results for programmers of parallel systems.

The renewal model of process behavior also predicts that process execution times in the presence of random delays asymptotically approach a normal distribution. We show using direct measurements that, in many programs, even relatively short execution intervals have distributions that are "close" to the normal, in a technical sense.

This paper is organized as follows. We define some key terms and concepts in Section 1.1 and review related work in Section 1.2. In Section 2 we present the renewal model and its analysis for variance as well as for distribution of task time. In Section 3, we present the results of our measurements of parallel programs. In Section 4, we discuss the implications of these results for the issues mentioned above. Finally, Section 5 summarizes the results of the paper, and suggests some interesting questions for future work.

1.1. Basic Terms and Concepts

Throughout this paper, we use the term *task* to denote a unit of allocation of work in a parallel program, and *process* to denote the logical entity that executes tasks. (The latter is sometimes called a *thread*). As an example, in a parallel loop with independent iterations and bounded by barrier synchronizations, the tasks would be the individual iterations of the loop and each process would typically execute a large number of such tasks between the successive barriers. By *synchronization point* we mean a partial or full barrier, i.e., a point in the execution of a process where it potentially has to wait for one or more other processes to complete some part of their execution. For the model in this paper, contention for software objects such as locks is a further source of random delays, rather than synchronization.

Finally, we emphasize that the focus of this work is the behavior of a parallel program (in particular, the variance and distribution of execution times) *for a single input data set*, rather than across different input sets. This is consistent with our goal of understanding the influence of non-determinism on synchronization costs within a parallel program.

1. The exception was a program that showed significantly higher execution times in some executions than in others, for the same input.

1.2. Related Work

To our knowledge, there has been no previous attempt to study the effect of random delays on the variance or distribution of execution times. However, one previous paper focuses on estimating the mean and variance of the processing requirements of tasks in the presence of data-dependent effects such as conditional branch probabilities and loop frequencies [21]. In that work, Sarkar describes a framework for determining the mean and variance of program execution times using frequency information from a counter-based execution profile of the program. His framework for estimating these parameters of execution times has been implemented as part of the PTRAN project at IBM Research [2]. However, he does not present any data from actual programs to show what parameter values occur in practice.

Finally, stochastic models that allow general distributions of task-time have been applied using different specific distributions, including the normal distribution. [6, 8, 10]. Dubois and Briggs [6] as well as Greenberg [8] argued that a task could be asymptotically normally distributed because it is the sum of a large number of (non-deterministic) instruction execution times. Our proof in Appendix A is essentially a formalization of this argument.

2. Renewal Model of the Effect of Random Delays

In this Section, we provide a framework for analyzing the variance and distribution of execution time attributable to random delays. We first describe a model of process behavior, and the assumptions in our analysis. In Section 2.1, we derive exact and approximate expressions for the variance, and use the exact expression to validate the approximation. We also use the approximate expression to study the variance for hypothetical values of the model parameters. In Section 2.2, we use the same model to derive the asymptotic distribution of execution time.

We consider a program executing on a parallel system, and focus on an interval in the execution of one process. In Section 3, we will apply the model to intervals between synchronization points, but the model and analysis in this section apply to other intervals such as a single task execution as well. Let D denote the total CPU requirement of the process in this interval, and let the random variable T denote the length of the interval, i.e., the total time to complete this processing requirement. In general, the execution of the process in the interval consists of a sequence of alternating processing and delay sub-intervals, where each delay represents a sub-interval in which the process busy-waits or is suspended (for example, for remote communication, access to a critical section, or other accesses to shared resources). Denoting the number of intervals required to complete the total processing requirement (D) by R, the lengths of successive processing sub-intervals by $\{P_i, i \geq 1\}$ and the lengths of delay sub-intervals by $\{C_i, i \geq 1\}$ with $C_\Sigma \equiv \sum_{i=1}^{R} C_i$, the total interval length is given by:

$$T = P_1 + C_1 + P_2 + C_2 + \cdots + P_R + C_R + P_{R+1} \tag{1}$$

We assume (1) that the $2 \times R$ random variables $\{P_i : 1 \leq i \leq R\}$ and $\{C_i : 1 \leq i \leq R\}$ are mutually independent, (2) $\{P_i : 1 \leq i \leq R\}$ have common distribution F_P, (3) $\{C_i : 1 \leq i \leq R\}$ have common distribution F_C, and (4) that each of these distributions has finite variance. In practice, in the presence of resource contention and non-stationary behavior (such as a burst of cache misses as each new task begins execution) these assumptions may be violated. One goal of the measurements in Section 3 is to validate the accuracy of the model results. Without these assumptions, the process behavior becomes much more difficult to analyze, and the solution would require more complex (and less intuitive) parameter values that would be difficult to measure for real programs.

For our further analysis of the model, we assume that D has zero variance, i.e., that D is constant for this program interval. This assumption is made because the goal of this analysis is to study the variability due to random delays alone. The random variable R and the sequence of processing times $\{P_i : 1 \leq i \leq R+1\}$ must satisfy $\sum_{i=1}^{R+1} P_i =$

D. Under these assumptions, denote the distribution, mean, variance and coefficient of variation of the total execution time T by $F_{T|D}$, $\mu_{T|D}$, $\sigma^2_{T|D}$ and $CV_{T|D}$, respectively.[2] Since D is assumed to be constant, $CV_{T|D} \equiv \sigma_{T|D}/\mu_{T|D} = \sigma_{C_\Sigma}/(D + \mu_{C_\Sigma})$. Thus, $CV_{T|D}$ is a measure of (normalized) variability in the execution time of the interval due to random delays. This is the principal measure we will use to understand the influence of random delays on specific programs, when we apply the model to these programs in Section 3.

If the assumption that D is constant is true for a particular program interval, then $CV_{T|D}$ is the coefficient of variation of execution time of that program interval. If the value of D can vary across different executions (for example, if the instructions executed in a task depend on which tasks have previously been completed by other processes), D itself can be influenced by the random delays experienced by this process as well as other processes of the program. However, if the variability of D is small relative to the average length of the interval, $CV_{T|D}$ can still be used to give a qualitative estimate of the impact of random delays. Thus, for such programs, it will be important to understand to what extent the value of D for this interval can vary across different executions of the program.

2.1. Analysis of the Normalized Variance $CV_{T|D}$

The main goal of this section is to derive an expression for $CV_{T|D}$ in terms of five input parameters: D, μ_P, σ^2_P, μ_C and σ^2_C, where the latter four terms denote the mean and variance of $\{P_i : 1 \le i \le R\}$ and the mean and variance of $\{C_i : 1 \le i \le R\}$ respectively. We first derive $\mu_{T|D}$ and $\sigma^2_{T|D}$ in terms of μ_R and σ^2_R, and then derive μ_R and σ^2_R in terms of the other input parameters. Note that R is determined only by $\{P_i\}$, i.e., it is independent of $\{C_i\}$. Hence, for $x \ge D$, we can write the distribution of T, $F_{T|D}(x) \equiv P\{T \le x\}$, as

$$F_{T|D}(x) = P\{R=0\} + \sum_{k=1}^{\infty} P\{R=k\} P\{\textstyle\sum_{i=1}^{i=k} C_i \le x - D\}, \quad x \ge D$$

$$= P\{R=0\} + \sum_{k=1}^{\infty} P\{R=k\} F_C^{(k^*)}(x-D), \quad x \ge D$$

where $F_C^{(k^*)}$ denotes the k-fold convolution of F_C with itself. (Note $F_{T|D}(x)=0$, $x<D$.) On taking transforms, we obtain:

$$\mathcal{F}_{T|D}(s) = \sum_{k=0}^{\infty} P\{R=k\} e^{-Ds} \mathcal{F}_C^{\,k}(s) \ . \tag{2}$$

Differentiating (2) and using $E[T^k] = (-1)^k \frac{d^k}{ds^k} \mathcal{F}_{T|D}(s)\big|_{s=0}$ gives $\mu_{T|D}$ and $\sigma^2_{T|D}$ in terms of μ_R and σ^2_R:

$$\mu_{T|D} = D + \mu_R \mu_C \tag{3a}$$

$$\sigma^2_{T|D} = \mu_R \sigma^2_C + \mu_C^2 \sigma^2_R \tag{3b}$$

We now focus on evaluating μ_R and σ^2_R, to derive both exact and approximate expressions for $\mu_{T|D}$ and $\sigma^2_{T|D}$.

2.1.1. Exact derivation for $\mu_{T|D}$ and $\sigma^2_{T|D}$

The analysis of R is complicated by the dependence between $\{P_i : 1 \le i \le R\}$ and P_{R+1}. We can avoid this difficulty by re-casting R as a function of a sequence of independent, identically distributed RVs $\{P_i' : 1 \le i < \infty\}$, each having the same distribution as each of $\{P_i : 1 \le i \le R\}$:

---

2. In general, we use μ_X, σ^2_X, CV_X and F_X to denote the mean, variance, coefficient of variation and distribution function of the random variable X ($CV_X \equiv \sigma_X/\mu_X$). We denote the Laplace transform of (the distribution of) X as $\mathcal{F}_X$.

$$R(t) \equiv \max\{r \geq 0 : \sum_{i=1}^{i=r} P_i' \leq t\} \tag{4}$$

Then R, as defined earlier, has the same distribution as $R(D)$, and $P_{R+1} = D - \sum_{i=1}^{R(D)} P_i$. But (4) is just the definition of a *renewal process* generated by $\{P_i' : 1 \leq i < \infty\}$; specifically, $R(t)$ is the number of renewals by time t [27].

The following expressions for $\mu_R(t) \equiv \mathrm{E}[R(t)]$ and its ordinary Laplace transform are well-known [27]:

$$\mu_R(t) = F_P(t) + \mu_R(t) * F_P(t),$$

$$\mathcal{L}(\mu_R(t)) = \frac{\mathcal{F}_P(s) / s}{1 - \mathcal{F}_P(s)}, \tag{5}$$

The corresponding expressions for $\mu_{R^2}(t) \equiv \mathrm{E}[R^2(t)]$ and its Laplace transform are also not difficult to derive [27]:

$$\mu_{R^2}(t) = \mu_R(t) + 2\int_0^t \mu_R(t-x)d\mu_R(x),\ t \geq 0$$

$$\mathcal{L}(\mu_{R^2}(t)) = \frac{(1 + \mathcal{F}_P(s))\ \mathcal{F}_P(s) / s}{(1 - \mathcal{F}_P(s))^2} \tag{6}$$

where $\mathcal{L}(\mu_R(t))$ and $\mathcal{L}(\mu_{R^2}(t))$ denote the ordinary Laplace transforms of $\mu_R(t)$ and $\mu_{R^2}(t)$ respectively, and $*$ denotes the convolution operator. (5) and (6) together allow us to compute $\mu_R(D)$ and $\sigma_R^2(D)$ for a particular distribution $F_P(t)$, but this generally requires numerical inversion of the Laplace transforms. A more practical application of (5) and (6) is to validate simpler approximations for μ_R and σ_R^2, which we derive next.

2.1.2. Approximate expressions for $\mu_{T|D}$ and $\sigma_{T|D}^2$

Estimates for $\mu_R(t)$ and $\sigma_R^2(t)$ are given in the Central Limit Theorem for renewal processes [27], which states that as $t \to \infty$, $R(t)$ is asymptotically normal with mean[3] t/μ_P and variance $t\sigma_P^2 / \mu_P^3$. Therefore, we estimate σ_R^2 by $D\sigma_P^2 / \mu_P^3 = (D/\mu_P)\mathrm{CV}_P^2$. Using these estimates for μ_R and σ_R^2 in (3) gives us our final approximate expressions for the mean and variance of T:

$$\mu_{T|D} \approx D\left(1 + \frac{\mu_C}{\mu_P}\right), \quad \sigma_{T|D}^2 \approx \frac{D}{\mu_P}\mu_C^2\left(\mathrm{CV}_C^2 + \mathrm{CV}_P^2\right), \tag{7a}$$

i.e,

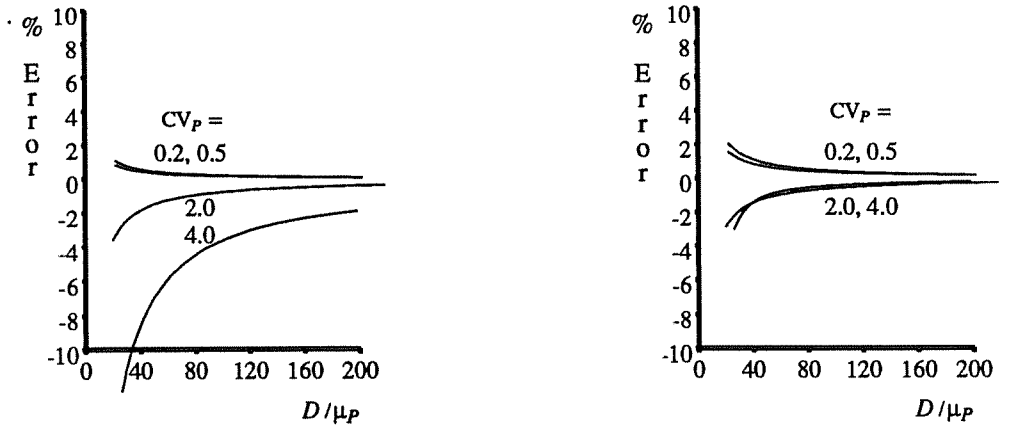
$$\mathrm{CV}_{T|D} \approx \frac{1}{\sqrt{D/\mu_P}} \frac{\mu_C}{\mu_C + \mu_P} \sqrt{\mathrm{CV}_C^2 + \mathrm{CV}_P^2} \tag{7b}$$

The key terms in this expression for $\mathrm{CV}_{T|D}$ include the average number of sub-intervals $D/\mu_P \approx \mu_R$, the average fraction of time the process is delayed $\mu_C/(\mu_C+\mu_P)$, and a term representing the variability of the individual processing and delay sub-intervals.

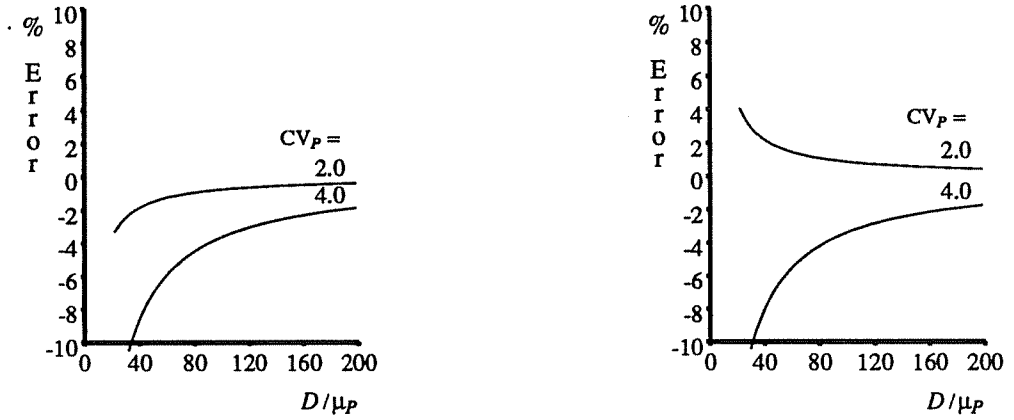
The above expressions for $\mu_{T|D}$ and $\sigma_{T|D}^2$ are only asymptotically exact, but they can be compared at finite D against exact values calculated using (5) and (6), for specific distributions F_P, and specific values of μ_P, μ_C, CV_P and CV_C. We do so in Figure 2.1 using gamma and 2-stage hyperexponential distributions for F_P, for $\mu_C/\mu_P = 1$, $\mathrm{CV}_C = 2.0$ and for a range of values of CV_P. The figure shows that for $\mu_C/\mu_P = 1$, the error in the approximation is potentially significant when $D/\mu_P < 40$ and CV_P or $\mathrm{CV}_C > 4$, but is likely to be acceptably low otherwise. The error increases when μ_C/μ_P increases (not shown), and $\mu_C = \mu_P$ is conservative relative to measured parameter values in Section 3.

---

3. In fact, this expression for the mean is exact for all $t > 0$ if, and only if, the first interval P_1 has distribution equal to the *distribution of residual life* of $P_i, i \geq 2$. For example, this holds automatically for the exponential distribution.



(a) F_P: Gamma Distribution ($\mu_P = 0.5$)



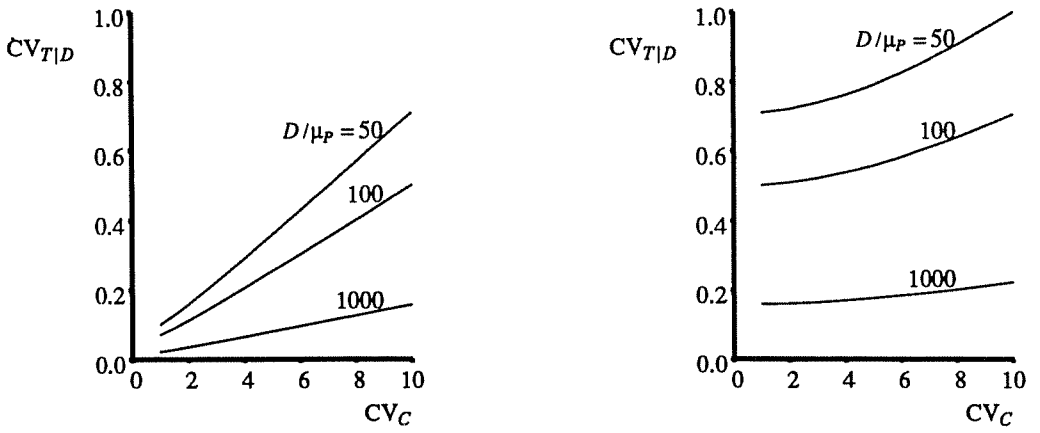
(b) F_P: Hyper-Exponential Distribution ($\mu_P = 0.5, p_1 = 0.9$)

Figure 2.1. Relative error in the approximations for Mean $\mu_{T|D}$ (left) and Variance $\sigma^2_{T|D}$ (right)

$\mu_C = \mu_P$, $CV_C = 2.0$

2.1.3. Quantifying the Variance due to Random Delays

A key observation from (7) is that $CV_{T|D}$ decreases as $1/\sqrt{\mu_R} = 1/\sqrt{D/\mu_P}$. Thus, for intervals containing a very large number of delays, we expect the total delay time to have very little overall variability relative to T. (Intuitively, the individual fluctuations of a large number of delay times will tend to cancel each other out in the long run.) We can use (7) to quantify how large D/μ_P must be for this observation to hold, for various values of the other parameters. In Figure 2.2, we again set $\mu_C = \mu_P$ and show $CV_{T|D}$ for a wide range of CV_P and CV_C. (Note that CV_P and CV_C are interchangeable as far as their influence on $CV_{T|D}$ is concerned.) The graphs show that $CV_{T|D}$ can be high (>0.5) when CV_P or CV_C is very high (≥10) and the interval contains 100 or fewer delays. For intervals with 1000 or more delays, however, $CV_{T|D}$ is low even when CV_P and CV_C are as large as 10 and $\mu_C = \mu_P$. Furthermore, when CV_P and CV_C are close to 1, even intervals containing as few as 50 delays have very low $CV_{T|D}$.



(a) $CV_P = 1$ (b) $CV_P = 10$

Figure 2.2. Effect of Random Delays on Relative Variability of Total Delay ($CV_{T|D}$)

$\mu_P = \mu_C = 1$

The above arguments are inconclusive about whether random delays cause significant variability in actual programs, since it is unknown what values of the model parameters occur in practice. One goal of the measurements presented in Section 3 is to obtain these parameter values for real programs, and show where in the parameter space typical programs can be expected to lie.

2.2. Asymptotic Analysis of the Distribution $F_{T|D}$

We were able to obtain explicit expressions for $\mu_{T|D}$ and $\sigma^2_{T|D}$ in terms of the Laplace transforms of μ_R and σ^2_R. Obtaining general expressions for the *distribution* of T is difficult. We can, however, derive the form of the distribution in the special case when D is large. In that case, we show in Appendix A that it is possible to apply the version of the Central Limit Theorem for cumulative (regenerative) processes [27] to prove:

$$T(D) \Longrightarrow \text{Normal}(\mu(D),\sigma(D)) \text{ as } D \to \infty,$$

$$\mu(D) = D + \frac{D\,\mu_C}{\mu_P},\quad \sigma^2(D) = \frac{D\sigma_C^2}{\mu_P} + \frac{\mu_C{}^2 D\sigma_P^2}{\mu_P{}^3} \tag{8}$$

where $\Longrightarrow$ denotes convergence in distribution.

Two points are worth noting here. First, the mean, $\mu(D)$, and variance, $\sigma^2(D)$ *are the same as* $\mu_{T|D}$ *and* $CV_{T|D}$ *calculated in (7)* using the estimate for μ_R and σ^2_R. Second, there is an important difference between the estimates for μ_R and σ^2_R and the asymptotic normal distribution of T calculated in (8): the convergence in (8) depends on σ^2_C, whereas the estimates of μ_R and σ^2_R did not. Thus, high variance of communication delays could make a moderately large task look different from normal, but the estimate for μ_R and σ^2_R, and hence the estimate in (7) for $\mu_{T|D}$ and $\sigma^2_{T|D}$, could still be accurate.

Table 3.1. Applications used for the Measurement Experiments.

Name	Application	Phase Structure	Dominant Phases	Input Data
MP3D	Hypersonic flow simulation	Five phases with intervening barriers	*Move* : more than 90% of total work	*Small*: 5000 molecules *Large*: 20000 molecules
Locus	Standard cell wire routing	Two iterations, parallel loop per iteration	-	*Small*: bnrE
Water	Water molecule simulation	One large, several small phases; intervening barriers (in each iteration)	*Inter-molecular force evaluation*: almost all the work.	*Small*: 64 mols *Large*: 343 mols
Barnes	Gravitational N-body simulation	One large, several small phases	*Force computation*: 90% of total work	*Small*: 1024 bodies *Large*: 8192 bodies
Bicon	Graph Biconnectivity	More than 50 phases; intervening barriers	-	*Large*: 4096 nodes, 16384 edges
Hydro	Microhydrodynamics: particle motion in viscous fluids	N parallel loops (N = no. of particles)	-	*Toy*: 2 particles *Small*: 8 particles
PSIM	Multistage network simulation	Single parallel phase per iteration	-	*Small*: 1024 node network *Large*: 4096 node network

3. Applications of the Model

In this section, we address the questions raised in Section 2, using data obtained from measurements of parallel programs. We begin with a description of the applications and measurement methodology in Section 3.1. In Section 3.2, we apply the renewal model to study the impact of random delays on these programs, by using the data to show where in the parameter space of the model these programs lie. In Section 3.3, we examine whether the assumptions of the model introduce significant errors into the qualitative conclusions obtained. While addressing this question, we measure the overall variance in process execution times, due to both random delays and variation in processing requirements. This also allows us to comment on the total variance found in practice. Finally, in Section 3.4 we examine whether real programs exhibit normally distributed process execution times in practice.

3.1. Applications and Measurement Methodology

We measured a variety of applications on a 20-processor Sequent Symmetry S-81, as well as a few shared-memory applications running on the Thinking Machines CM-5. Table 3.1 gives a brief overview of the applications and inputs (*Small* and *Large* are mainly labels we will use for convenience. The *Large* input size is a somewhat more realistic data set than the *Small* size). Four of the applications (MP3D, Locus Route, Water and Barnes) are from the Splash suite, which was developed to provide a realistic set of parallel applications for performance evaluation of parallel systems [23]. The other three are also real applications in the sense that they were written to solve computationally intensive problems of interest to their authors. Hydro is a parallel simulation of particle motion in viscous fluids, with efficient communication. [7]. PSIM was developed at Lawrence Livermore Laboratories to simulate the indirect binary n-cube memory server network in a large parallel vector-processing environment [4]. Bicon is an implementation of a parallel algorithm to find the biconnected components of large graphs [24].

We measured each of the programs running stand-alone, allowing us to characterize the non-determinism intrinsic in the program. This is important because parallel system models (such as those for parallel program performance prediction or for analysis of scheduling policies) require parameters that characterize the intrinsic behavior of the program as input. It is also worth noting that intrinsic random delays are the key unknown for determining synchronization costs in multiprogrammed systems where the processes of an application are (essentially always) co-

scheduled. The intrinsic random delays in the applications of Table 3.1 are communication delays due to remote memory accesses. In particular, page faults and lock contention did not cause significant delays in these programs.

For each application, we focused on one to three phases of the program, where a phase is bounded by barrier synchronizations, and has no intervening synchronization points. We measured one process in each phase, and the interval length T corresponds to the execution time of the measured process between the corresponding barriers. D is the total processing in that interval, and the delay sub-intervals $\{c_i\}$ are the remote communication delays. Some experiments require repeated measurements of a particular phase; exactly the same input data set is used for each such measurement, as explained in Section 1.1.

3.2. Measurements for Evaluating Execution Time Variance

Before presenting the results for the evaluation of variance due to random delays, it is important to note that $CV_{T|D}$ (the measure of interest) will be estimated for a program phase from a single execution of the program. As discussed in Section 2, for program phases in which the processing requirement (D) does not vary across different runs, $CV^2_{T|D}$ is the normalized variance in execution time of the interval, due to random delays. For phases in which D varies across runs but the variability in D is small relative to the average length of the interval (see Section 3.3), $CV_{T|D}$ should still give a qualitative estimate of the influence of random delays on the variance of execution time.

3.2.1. Measurements on the Sequent Symmetry

Shared-memory on the Sequent Symmetry is supported by an invalidation-based snooping cache protocol, and the communication delays are due to three types of remote requests (*read-shared*, *read-invalidate* and *invalidate*). The measurements were made using non-intrusive hardware probes to record the relevant bus events.[4] The various parameter values were subsequently derived from the stored traces.

Table 3.2 gives the measured parameter values, as well as the model results ($\mu_{T|D}$ and $CV_{T|D}$), for each of the application phases executing on 16 processors on the Sequent. Values for μ_P and μ_C are given in units of bus cycles (equal to 0.1 microseconds), and for D and $\mu_{T|D}$ in units of 1000 bus cycles (100 microseconds). The applications are listed in generally increasing order of processing demand D, although the demand varies from phase to phase in each application. In addition to the five basic input parameters, we also give the values of $R = D/\mu_P$ (the measured number of random delays) and $\mu_C/(\mu_C+\mu_P)$.

Before interpreting the results obtained, two points are worth noting about the measured parameters in Table 3.2. First, the values fall in the region of the parameter space where the approximate solution of the renewal model is expected to be accurate (Figure 2.1). Second, the measured application phases vary widely in terms of all five input parameters, as well as the number of random delays (D/μ_P) and communication overhead ($\mu_C/(\mu_C+\mu_P)$). In particular, the total processing demand (D) varies by more than 4 orders of magnitude (from 770 microseconds for the *Res–Move* phase of MP3D to more than 33 seconds for Hydro), the number of random delays varies (somewhat in proportion to D) from 33 to 72707, CV_P and CV_C are as high as 4.35 and 1.68, respectively, and communication overhead is as high as 0.24 (Bicon). Thus, the measured applications have widely varying behavior, and communication requests experience significant variability.

The most striking observation from the table is that in every one of these application phases, the predicted variability of execution time due to random delays ($CV_{T|D}$) is extremely low. The highest estimated $CV_{T|D}$ on this system is 0.022 (for the *Res-Move* phase of MP3D) and this was obtained with a much smaller input size than expected in

4. A Tektronix DAS 9200 Logic Analyzer was used for this purpose.

Table 3.2. Measurements of Renewal Model Parameters: Sequent Symmetry.

Application			Measured Parameter Values							Model Results	
Program	Phase	Input	D ($\times 10^3$)	μ_P	μ_C	CV_P	CV_C	$R = \frac{D}{\mu_P}$	$\frac{\mu_C}{\mu_P+\mu_C}$	$\mu_{T\|D}$ ($\times 10^3$)	$CV_{T\|D}$
MP3D	*Res-Move*	Small	7.7	236.9	12.00	2.00	1.68	33	0.05	8.0	0.0221
	Res-Coll	Small	11.7	182.0	9.90	1.38	1.30	64	0.05	12.4	0.0122
	Move	Small	608.2	410.1	14.6	1.03	0.80	1483	0.03	629.8	0.0012
	Res-Move	Large	26.3	316.5	11.60	1.34	1.49	83	0.04	27.2	0.0078
	Res-Coll	Large	33.7	267.8	11.30	1.13	1.26	126	0.04	35.1	0.0061
	Move	Large	2396.7	349.0	12.9	1.06	0.43	6867	0.04	2485.1	0.0005
PSIM	-	Small	325.5	76.2	11.1	0.86	0.50	4272	0.13	372.8	0.0019
	-	Large	1507.9	66.8	10.8	0.74	0.44	22573	0.14	1752.2	0.0008
Bicon	*Conn1*	Large	191.7	147.5	23.5	2.23	0.79	1300	0.14	222.3	0.0090
	Lowhigh	Large	279.9	217.4	8.7	1.15	0.51	1287	0.04	291.1	0.0014
	Tour	Large	3785.2	72.7	23.4	4.82	0.67	52066	0.24	5003.54	0.0053
Locus	*Itn. 2*	Small	2159.8	231.4	10.60	3.58	0.53	9334	0.04	2258.9	0.0016
	Itn. 1	Small	2904.1	137.9	14.70	4.02	0.80	21059	0.10	3214.1	0.0027
Water	*Inter-mol*	Small	4203.0	1584.8	9.9	3.71	0.36	2652	0.006	4229.1	0.0004
	Inter-mol	Large	122667.6	3767.5	10.50	2.55	0.34	32559	0.003	123009.4	0.0000
Barnes	*Force*	Small	16577.9	4781.6	8.0	2.42	0.34	3467	0.002	16605.6	0.0001
	Force	Large	223443.5	3073.2	7.6	2.22	0.26	72707	0.002	223995.2	0.0000
Hydro	*Nucleus*	Toy	102967.3	9662.8	8.5	4.35	0.36	10656	0.001	103057.8	0.0000
	Nucleus	Small	336718.3	6735.9	8.0	2.73	0.33	49989	0.001	337120.0	0.0000

practice [23]. To analyze these results further, we compare the measured parameters against regions of the parameter space in Figure 2.2 (recalling that Figure 2.2 is pessimistic for communication overhead less than or equal to 0.24). In all but a few of the cases $D/\mu_P > 1000$ and, as shown in Figure 2.2, this alone explains why the predicted $CV_{T|D}$ is extremely low for those cases. Thus, D/μ_P has a dominant role in determining $CV_{T|D}$ in most cases on this system. In the few cases where D/μ_P is on the order of 100 or less (and in many other cases in the Table), CV_P and CV_C are both less than 2. Thus, again, $CV_{T|D}$ is extremely low. In fact, for the worst case combination of parameter values across all the measured application phases ($D/\mu_P = 33$, $\mu_C/(\mu_C+\mu_P) = 0.24$, $CV_P = 4.82$ and $CV_C = 1.68$), the renewal model predicts that $CV_{T|D}$ would be 0.21, which is still low.

We next consider how these parameter values could change for applications on highly parallel shared-memory systems. First, the range of D/μ_P seen here (more than 4 orders of magnitude) could be representative of larger systems as well. Large values of D/μ_P are still likely to occur due to scaling up problem sizes, yet the fraction of applications that have small D/μ_P could also increase if finer granularity of synchronization is supported efficiently in future systems, or if applications can use new primitives to completely overlap a significant fraction of communication events with computation. In applications for which D/μ_P is as large as in most cases seen here, $CV_{T|D}$ should continue to be low, because the effect of $\mu_C/(\mu_C+\mu_P)$, CV_P and CV_C would have to be one to two orders of magnitude higher than observed here to yield even $CV_{T|D} \geq 0.1$. In applications where D/μ_P is significantly lower, increases in $\mu_C/(\mu_C+\mu_P)$, CV_P or CV_C are important to consider. It is not clear how these parameters would extrapolate to other applications and systems. efficiency considerations alone dictate that $\mu_C/(\mu_C+\mu_P)$ will generally be less than about 0.5 (i.e., 50% efficiency).[5] CV_P arises due to the non-uniform intervals between cache misses and there do not appear

5. On multithreaded processors, however, $\mu_C/(\mu_C+\mu_P)$ could be somewhat higher and still allow reasonable efficiencies.

Table 3.3. Measurements of Renewal Model Parameters: Sequent Symmetry with Bus Load.

Application			Measured Parameter Values							Model Results	
Program	Phase	Input	D ($\times 10^3$))	μ_P	μ_C	CV_P	CV_C	$R = \frac{D}{\mu_P}$	$\frac{\mu_C}{\mu_P+\mu_C}$	$\mu_{T\|D}$ ($\times 10^3$))	$CV_{T\|D}$
MP3D	*Res-Move*	Small	26.3	315.7	45.80	1.66	0.75	83	0.13	30.1	0.0253
	Move	Small	2342.0	408.7	59.40	0.99	0.67	5730	0.13	2682.6	0.0020
PSIM	-	Small	1303.9	71.9	46.1	0.76	0.82	18134	0.39	2139.8	0.0032
Bicon	*Conn1*	Small	162.8	89.7	49.80	1.84	0.70	1815	0.36	253.2	0.0165
	Tour	Small	6443.5	81.1	54.1	0.94	0.68	79451	0.40	10741.8	0.0016

to be reasons to believe that this will be significantly different in future systems. We next discuss two further experiments that were designed with the goal of producing higher values of CV_C and $\mu_C/(\mu_C+\mu_P)$, which could exist in future parallel systems. In any case, the model demonstrates the need to obtain these parameter values, and provides a framework for estimating the influence of these parameters as future systems become available.

3.2.2. Measurements on the Sequent with External Bus Load

For this experiment, we wrote an artificial parallel program to increase the bus contention. All tasks of the "bus-loading" program repeatedly read and write a fixed memory location causing that cache line to bounce from cache to cache. We repeated the measurements with MP3D, PSIM or Bicon running on 4 processors, and the bus loading program on 14 other processors. (We used the smaller input size in each case, which gives approximately the same total work per process per phase as the larger input on 16 processors). The results for these measurements are given in Table 3.3. Comparing with the numbers in Table 3.2, we see that $\mu_C/(\mu_C+\mu_P)$ is much higher, yet CV_C is not significantly different despite the increased contention. Thus, the highest estimated value of $CV_{T|D}$ seen here is still only 0.025.

3.2.3. Measurements on the CM-5

We were also able to measure two of the above programs on a 32-processor Thinking Machines CM-5, a system that is scalable to much larger numbers of processors. Since the CM-5 does not support shared memory, we used the Wisconsin Windtunnel [20] to simulate the execution of the shared-memory applications. In this simulator, the application program executes at full hardware speed on the processing nodes of the CM-5, but traps into software on memory references to cache blocks that would not be in the target machine's cache. Explicit messages are used to obtain remote cache blocks. As on the Sequent, the delays for remote communication were the delay intervals, $\{C_i : 1 \le i \le R\}$. Since every remote communication event causes a trap into the software handler on the local node, the measurements are done in software. In measuring the processing intervals, we ensured that the time to service remote memory requests (i.e. interrupts) from other nodes was excluded. Other than these interrupts the application runs at full hardware speed between traps; hence the measured processing intervals are realistic values for these applications. The software overhead required to service the traps could not be fully eliminated and therefore the measured communication costs are higher than on large-scale parallel systems of the future. These measurements serve to test our conclusions under high communication overhead.

We ran the simulator with a 64 kilobyte, 4-way set associative cache per node, and a full-directory non-broadcast invalidate cache coherence protocol [1]. One might expect CV_C to be high on this system because some but not all remote requests have to be forwarded from the directory to a third node that will supply the updated copy of the block, and there could be significant queueing delays for the trap handlers on the nodes.

Table 3.4. Measurements of Renewal Model Parameters: CM-5 + Wisconsin Windtunnel.

Application			Measured Parameter Values							Model Results	
Program	Phase	Input	D (ms)	μ_P (μs)	μ_C (μs)	CV_P	CV_C	$R = \frac{D}{\mu_P}$	$\frac{\mu_C}{\mu_P+\mu_C}$	$\mu_{T\|D}$ (ms)	$CV_{T\|D}$
MP3D	*Move*	Small	44.82	49.3	1551.5	6.18	0.88	909	0.969	1454.86	0.2006
	Move	Large	147.84	42.0	1508.5	3.68	0.76	3520	0.973	5451.10	0.0617
Water	*Inter-mol*	Small	253.8	564.0	794.8	4.12	1.75	450	0.585	611.5	0.1234
	Inter-mol	Large(512)	6729.27	376.8	771.9	3.26	0.50	17859	0.672	20514.63	0.0166

The data for MP3D and Water are given in Table 3.4. The communication overhead for Water is high (as expected) but probably only slightly higher than is likely in communication-bound applications on future parallel systems. In MP3D however, the overhead is extremely high, primarily because of very frequent remote communication. In practice, applications may have to restructured for less frequent communication, and MP3D can be considered an extreme case to test the conclusions of the model. Despite the high overhead, CV_C is *not significantly higher* than in the previous experiments for either program, and CV_T is still only 0.2 or less. Unless much higher CV_C is observed when more of the communication is implemented in hardware, it seems plausible that even in such systems communication costs will not introduce significant variability in execution times.

3.3. Measurements of CV_T (and $CV_{T|D}$)

The measurements we present in this subsection have two goals. First, we estimate CV_T, i.e, the variability in T due to all sources, using direct measurements of a set of the application phases, to see how much overall variability occurs in practice. Second, we use these measurements of CV_T (1) to qualitatively validate the values of $CV_{T|D}$ predicted by the renewal model in cases where this is possible, and (2) to compare the relative influence of variability in processing requirement D and variability in total delay C_Σ in cases where this is relevant.

Table 3.5 gives values of CV_T measured in software for several application phases on the Sequent Symmetry (estimated using samples of T from 100 to 300 runs of each phase).[6] The predicted values of $CV_{T|D}$ for these phases are repeated from Tables 3.2 and 3.3 to aid in the comparisons below. The measured values of CV_T show that in all but one of these programs, the overall variability due to processing requirements as well as communication delays is very low (less than 0.05), and it is also fairly low in the exceptional case of Locus Route.

For several of the measured phases (i.e, those in MP3D, Water and *Barnes*), D is fixed for different runs on the same input. In these cases, CV_T should be equal to $CV_{T|D}$, thus the measured CV_T can be directly used to validate the predicted values of $CV_{T|D}$. In each of these cases, the measured and predicted values agree within the expected accuracy of the software measurements. For the remaining cases in the table, the measured CV_T provides an (approximate) upper bound on $CV_{T|D}$. Thus, in all cases except Locus Route the measured values of CV_T directly indicate that the very low variability predicted by the renewal model is qualitatively correct.

Bicon, Locus Route and PSIM each have variability in D due to the nature of the computational algorithm. (For example in Locus Route, a process repeatedly computes the route for the next wire in a work queue, where the time to compute the route depends on which wires have previously been routed. Variability in the initial task assignments as well as in the *individual task* execution times due to random delays will cause D to vary across different

6. A higher degree of accuracy in the measured CV_T could be obtained with increased effort by increasing the number of samples and/or by using hardware measurement probes. The increased accuracy is not needed here.

Figure 3.5. Measured Total CV_T and Predicted $CV_{T|D}$ †

Program	Phase	Input	Measured	Predicted
MP3D	*Move*	Small	0.0076	0.0020
MP3D	*Move*	Large	0.0058	0.0005
PSIM	-	Small	0.0335	0.0019
PSIM	-	Large	0.0098	0.0008
Bicon	*Conn1*	Large	0.0108	0.009
Bicon	*Tour*	Large	0.0345	0.0053
Bicon	*Lowhigh*	Large	0.0404	0.0014
Locus	*Itn. 2*	Small	0.0882	0.0016
Locus	*Itn. 1*	Small	0.1396	0.0027
Water	*Inter-mol*	Small	0.0006	0.0004
Barnes	*Force*	Small	0.0008	0.0001
Barnes	*Force*	Large	0.0005	0.0000

† First case is for MP3D running on 4 processors, with bus load from 14 processors. All others are on 16 processors, with no external load.

runs.) In these cases, the values of CV_T and $CV_{T|D}$ (assuming these are approximately accurate) suggest that the variability in D dominates that in the total delay (but is still low). In the case of Bicon, we were able to obtain further evidence for this conjecture, using the following calculation for the *Lowhigh* phase. Each task execution time in this phase is determined by a measure of a node in the graph that appears to vary quite randomly for each task across different runs. Furthermore, the measured variance of execution times of all tasks of the phase in a single run on one processor agreed very closely with the measured variance of a subset of the tasks that would have been assigned to the measured process during a parallel run. We thus hypothesized that the meas- ured variance in task times on a single processor is a good estimate of the variance in task processing requirements across different parallel runs, for the tasks assigned to the process. Using this to calculate the component of CV_T^2 due to variations in D, we obtained a value of $(0.042)^2$. Thus, this value added to $CV_{T|D}^2$ agrees quite closely with the measured value of CV_T^2.

3.4. Measurements of Execution Time Distribution

In Section 2.2, we showed that the distribution of execution time in an interval asymptotically approaches a normal distribution, when the non-determinism is due to random delays. Since this is an asymptotic result and since there are other sources of non-determinism, it is important to determine if real program phases show normally distributed execution times. This can be done using the measured samples of running times used in Section 3.3. Here, the samples are used to construct an empirical distribution which is an estimate for the unknown parent distribution. Furthermore, the mean μ and variance σ^2 of the samples can be used to construct a Normal distribution, and thus the shape of the estimated parent can be directly compared to a Normal for each phase. In addition, the Kolmogorov-Smirnov statistic can be constructed from the measured samples and used to derive a *confidence band* for the actual parent distribution [26]. This gives an error bound between the estimated and actual parent distribution at a certain level of confidence.

The empirical distribution calculated using 300 samples, the upper and lower ends of the 95% confidence band, and the predicted Normal distribution, are shown in Figure 3.1(a) for the *Conn1* phase of Bicon. We see that the empirical distribution very closely tracks the normal distribution. The corresponding curves are given for the *Lowhigh* phase in Figure 3.1(b), and although we believe the variations in task execution times are the dominant cause of variance in this case, we again see that the empirical distribution closely tracks the normal. In fact, this is not surprising

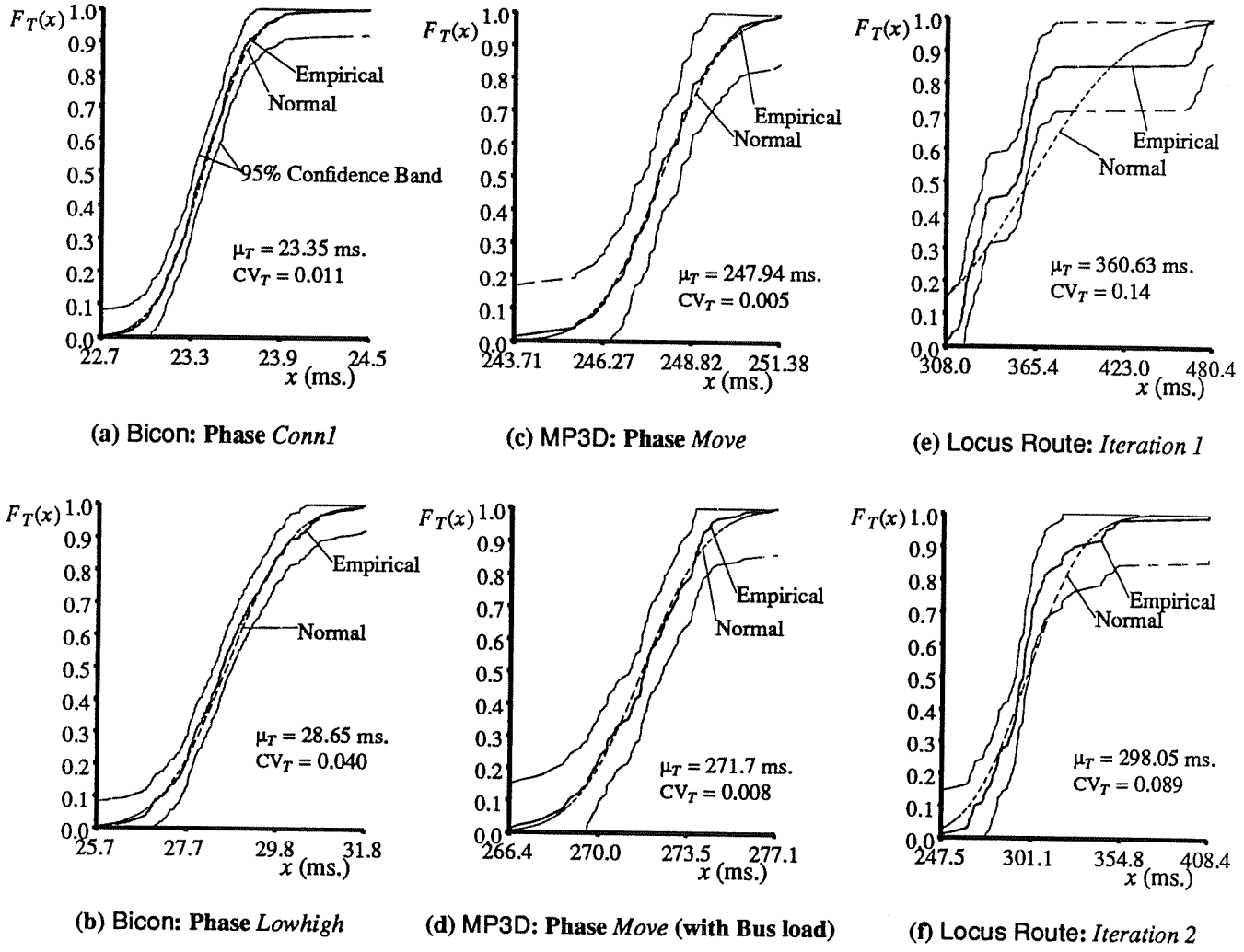


Figure 3.1. Comparing Measured Execution Time Distributions and Predicted Normal for Various Program Phases

because the measured process executes 256 individual statistically identical tasks, and thus the sum of the processing requirements itself has converged to a normal distribution. The width of the confidence bands is ±0.076 in both cases. The same data (from 80 and 100 samples) are shown in Figure 3.1(c) for the *Move* phase of MP3D executing on 16 processors with an input size of 20000 particles, and in Figure 3.1(d) for MP3D executing on 4 processors with input size of 5000 particles, but with the external bus loading program executing on 14 processors in the latter case. In both cases, the distribution is very close to normal.

Finally, the curves for the two iterations of Locus Route are shown in Figure 3.1(e,f). The empirical distributions (calculated using 100 samples each) are different from the corresponding (predicted) normal distribution in each case, particularly for Iteration 1 which clearly shows a trimodal form. The processing demands of a task in this program can be affected by which tasks have been completed previously (and thus even by the order of previous completions), and in a few runs the execution times of the iterations are significantly higher than in most other runs. The variability in communication delay, which would otherwise yield normally distributed execution times, is much smaller than this variability in processing demand.

4. Implications of the results

The data presented in Section 3 strongly indicate that the principal effect of communication delays in shared-memory parallel programs is to increase the mean completion time of a process in a phase, and not the variance, even under conditions of high communication costs and contention. Furthermore, the data indicate that the overall variance of execution time between synchronization points due to both communication delays and processing requirements is also extremely small in many programs. In this Section, we use examples to demonstrate the implications of the above results for parallel program performance prediction, as well as for other performance models of parallel systems. We also briefly touch upon the implications of the above results for parallel programming, as well as the implications of the results presented in Section 3.4 that moderately large processes in many programs have normally distributed task execution times.

4.1. Implications for Parallel Program Performance Prediction Models

A fundamental step in parallel program performance prediction is to estimate the mean waiting time experienced by processes at synchronization points. Calculating synchronization delays in a model based on non-deterministic task times is, however, extremely difficult. Thus, stochastic models that apply to any but the simplest program structures have had to assume exponentially distributed task execution times for tractability, and yet require extremely complex solution heuristics [9, 15] or Markov chains with exponentially growing state space sizes [16, 25]. The results of our work indicate that models that assume exponentially distributed task execution times may seriously overestimate the waiting time for many programs. The renewal model shows that it is possible for very small tasks to experience significant variance in execution time due to random delays. In such cases the assumption of exponentially distributed task times could be accurate. The data in Section 3 show, however, that such tasks have to be extremely small; for example, even tasks of a few hundred instructions may not match the exponential task assumption. Thus, for an analytical model to be accurate for many programs, it should be able to represent tasks with low or extremely low variance. To date, stochastic models with this capability have been restricted to programs with very simple task graphs and require further simplifying assumptions such as i.i.d. (independent, identically distributed) tasks within each parallel phase [10, 14].

The results of Section 3 suggest a different and potentially simple approach to this problem. Specifically, those results indicate that *ignoring the variability* in task times when evaluating the waiting time at synchronization points could give accurate results in many programs. With this assumption, estimating synchronization delays should be relatively simple for a large class of programs, once the mean execution times of the tasks (including mean communication and other delays) are known.

To further evaluate these implications, we present some preliminary data comparing the results obtained from three performance models for the MP3D application on the Sequent Symmetry. The first two models are specific instances of the Kruskal and Weiss model [10]. This model is restricted to programs with alternating serial and parallel phases where each parallel phase consists of i.i.d. tasks, but allows any continuous distribution of task execution time with coefficient of variation less than or equal to 1. In the first case, we apply this model assuming exponentially distributed task execution times. Note that any errors introduced by this assumption will be found in the other models that assume exponentially distributed task times as well [9, 15, 16, 19, 25]. Henceforth, we refer to this as the Exponential Task model. In the second case, we apply the Kruskal and Weiss model using the true variance of task times (actually, estimates of the variance obtained as explained below). The third model is a simple deterministic model as suggested above.

The parameter values for these models were derived as follows. The individual task processing requirements were measured in software with the program executing stand-alone on a single processor (to eliminate contention

Table 4.1. Results of Performance Prediction Models for MP3D.

Input Size	Procs	Measured Time (sec)	Exponential Model		Actual Variance Model		Deterministic Model	
			Prediction	%Error	Prediction	%Error	Prediction	%Error
5000 particles	1	0.998	1.015	1.722	1.015	1.722	1.017	1.926
	2	0.508	0.551	8.584	0.512	0.793	0.514	1.126
	4	0.258	0.317	22.501	0.260	0.738	0.260	0.527
	8	0.134	0.193	44.218	0.133	-0.505	0.133	-0.483
	12	0.093	0.150	61.239	0.091	-1.948	0.092	-1.606
	16	0.073	0.126	71.762	0.069	-5.859	0.069	-6.264
20000 particles	1	3.891	4.009	3.039	4.009	3.039	4.005	2.924
	2	1.980	2.089	5.468	2.011	1.570	2.007	1.336
	4	1.002	1.123	12.019	1.014	1.105	1.009	0.688
	8	0.509	0.631	23.938	0.514	1.017	0.510	0.173
	12	0.345	0.462	33.770	0.348	0.642	0.342	-0.940
	16	0.267	0.374	39.844	0.263	-1.675	0.258	-3.601

effects). The mean communication cost for each phase on P processors was estimated using a simple M/M/1//P model for the bus. For each phase, the request rate used for all values of P was the mean request rate $1/\mu_P$ measured with the program running on 16 processors (the values given in Table 3.2). Although this introduces some error in the predicted remote memory response times when $P<16$, it should affect all models approximately equally. We assumed there was no queueing for the caches or memory. The only other data required are the actual variance of task execution times, for the second Kruskal and Weiss model alone. Since the model assumes i.i.d. tasks per phase, the (small) variations in processing requirements across the different tasks in a phase were incorporated as one component of the variance. This was added to the variance of communication delays in each task, estimated using the analytical model in Section 2 and using the parameter values presented in Section 3.2 (except that D now corresponds to the average processing requirement of the individual tasks rather than the total for the entire phase). It is worth noting that, to the best of our knowledge, this is the first application of the Kruskal and Weiss model for a real program, and the renewal model for the variance due to random delays was crucial in making this possible.

The measured running times of the program for 5000 and 20000 particles (averaged across 40 runs in each case), the corresponding running times predicted by the three models, and the percentage error in these predictions are given in Table 4.1. The exponential task model is pessimistic for even small values of P: as expected, it greatly overestimates the synchronization cost at each barrier. The error is appreciably lower (but still quite large) in the case of the larger input size because there are many more tasks per phase, which mitigates the influence of the exponential task assumption to a greater extent. (For example, the dominant *Move* phase has 317 and 1,274 tasks for the small and large input sizes respectively.) The Kruskal-Weiss model using the actual variance of task times is extremely accurate since the program fits the assumptions of the model well and the model accurately accounts for all the significant factors (the mean and variance of task processing times, as well as the mean and variance of communication delays). The Deterministic model is almost as accurate in all cases even though it ignores the variance due to communication. Both these models have errors of only 1% to 3% in all but one case (which showed about 6% error).[7]

7. One might expect the predicted execution times on 1 processor to be almost exactly equal to the measured value. However, the measured total running times given in the table are averages across 40 runs, whereas the individual task processing times used by the models are obtained from a single measurement run. This introduces a small discrepancy.

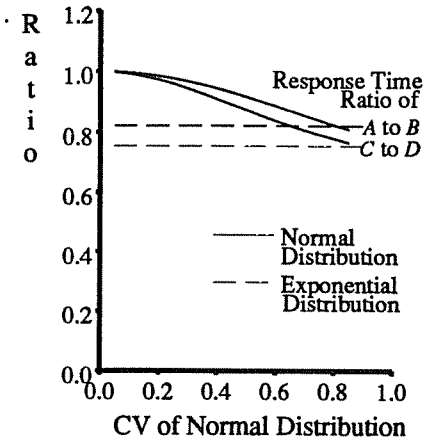
Although this is only one application on one system, it serves to demonstrate the potential significance of the result that execution time variance may be very low for many programs. The deterministic model has the advantage that it does not require the variance due to communication delays, which is much more difficult to estimate than the mean delay alone. It also has the potential advantage that it might more easily be extended to more general task graphs than the Kruskal and Weiss model. A more complete evaluation of the deterministic approach for performance prediction is a subject of further study.

4.2. Implications for General Parallel Processing Models

A number of stochastic models have also been used to study various aspects of parallel systems [3, 5, 11-13, 17-19, 22, 28]. In many cases, the goal of these models is a qualitative comparison of design alternatives based on quantitative performance estimates (for example, models used to compare scheduling policies). Many of these models assume exponentially distributed task or process execution times to permit tractable analysis. Our results show that it is important to determine by informal reasoning or formal validation how the model results depend on the exponential assumption. If a result would be significantly weaker, or even different, for programs that have much lower task time variance, it may not apply to many parallel programs.

One result that is not strongly dependent on the distribution assumption is as follows. Nelson et al [17] showed that in an environment containing mixed sequential (interactive) and large parallel (batch) jobs, an unpartitioned system yields better performance than one in which processors are statically partitioned among the two classes. They assumed that the parallel jobs consisted of tasks with exponentially distributed execution times. But, in fact, Setia and Tripathi [22] showed that the same conclusion holds with a completely different assumption about task times. Specifically, they assumed that *each job consisted of tasks of equal size*, while the total execution time of the jobs on any fixed number of processors was assumed to be exponentially distributed.

In contrast, one result that is significant for exponentially distributed tasks but weaker for tasks with lower variance is Nelson's result [19] that higher variance of parallelism can yield lower response times, when queueing effects are small. He considers a parallel processing system with P processors and an arrival stream of parallel jobs, where an arriving job splits into n tasks with probability β_n, $1 \le n \le N_{max}$. Each task has exponentially distributed processing requirement with unit mean. He compares two types of jobs, one with higher variance of parallelism than the other, and shows that the jobs with higher variance have significantly lower response times when queueing effects are negligible. This result arises because the completion time of n tasks on P processors is a concave increasing function of n. (For example, for exponentially distributed task times with mean $1/\mu$, and for $n \le P$, it is given by $(1/\mu) \sum_{i=1}^{i=n} 1/i$.) Although this result holds for any concave task time distribution, it is weaker when the curve grows more slowly as a function of n, such as for distributions with lower variability. To show the effect of the choice of distribution, consider the two systems (A and B) that were compared in [19], each with 8 processors but different distributions of parallelism (β_i) as given in Figure 4.1. Both systems have mean parallelism of 2.4, but A has a higher variance of parallelism than B. If each task has exponentially distributed execution time, the ratio of the mean response times of A and B is about 0.82 (i.e., A has 18% lower response time). Now consider the same two systems, but assume task execution times are normally distributed with unit mean and variance σ^2. In Figure 4.1, we plot the ratio of average response times of A and B for a range of values of CV = σ. The ratio is close to 1 for low variance and approaches approximately 0.82 as σ gets close to 1. We repeat this comparison for another pair of systems, C and D, with 32 processors each, and C having a higher variance of parallelism. The performance ratio of C to D is about 0.75 with exponentially distributed task times, but it is again close to 1 for low σ. Thus, jobs with higher variance of parallelism do show lower response times, but the effect is significant only when the variance of task execution time is high.



Class	Dist.	Mean $\bar{\beta}$
A	$\beta_1 = 0.8$ $\beta_8 = 0.2$	2.4
B	$\beta_2 = 0.6$ $\beta_3 = 0.4$	2.4
C	$\beta_1 = 0.5$ $\beta_{32} = 0.5$	16.5
D	$\beta_{15} = 0.5$ $\beta_{16} = 0.5$	16.5

Figure 4.1. Comparing predictions from Exponential and Normal Distributions.

The above example is intended only to emphasize that it is important to evaluate the effects of distribution assumptions on the results of a model, as we stated earlier. If a result is dependent on a task time distribution with high variance, our findings show that it may not apply to many parallel programs.

4.3. Implications for Programmers

Load balancing is an important aspect of the design of efficient parallel programs. In particular, programmers have to choose between static and dynamic load balancing for probably every parallel program. Static load balancing is an attractive choice because it is simpler to implement and debug. In most systems today, static scheduling performs well if the computation load is evenly balanced. In future systems with much higher communication costs relative to computational speed, static scheduling would not perform well if contention introduced significant variability in task completion times, even when the computational work is divided evenly among the processes. Our results indicate that this problem is not likely to arise in practice, and the effect of random communication delays can usually be ignored in making the choice between static and dynamic scheduling.

4.4. Implications of Normally Distributed Task Times

Knowledge of the shape of the distribution of task times, while interesting in itself, may also have some practical implications. For example, in simulation studies of parallel systems, some assumptions about the workload are necessary. In particular, when non-deterministic tasks are modeled, some default distribution is often chosen. Our results show that a normal distribution should be a reasonable choice in many cases.

5. Conclusions and Future Work

In this paper, we have studied the effect of random delays, as well as other sources of non-determinism, on the execution time of processes in parallel programs. We described an analytical model of program behavior that yields considerable insight into the effect of random delays on the variance and distribution of process execution time over any interval. We used detailed measurements of several shared-memory programs on two different systems to parameterize and apply the model to those programs, and thus to evaluate the variance of process execution time between synchronization points due to communication delays. We also used direct measurements of variance due to all sources of non-determinism. The key conclusions of our study, for shared-memory programs on systems of the

foreseeable future, are:

- Communication delays introduce negligible variance into the execution time of a process between successive synchronization points, even under conditions of high communication cost and contention. Furthermore, this conclusion is likely to hold for task execution times of moderate or large size as well.
- For many but not all such programs, non-deterministic processing requirements also introduce very little variance into the execution time between synchronization points. (In particular, heuristic search programs can be exceptions.)

We then showed that the above results could have potentially important implications for performance analysis of parallel programs and systems. One consequence of these results is that a deterministic model might be used for parallel program performance prediction, because it appears reasonable to ignore the variability in execution times when estimating synchronization delays (as we showed for one example program). This could be important because a deterministic model may be much simpler to solve than stochastic models, and may apply to a large class of programs as well.

A related consequence of our results is that the assumption of exponential task times can produce large errors in running time estimates. Again, this argument is borne out by the example program mentioned above, and underscores the potential advantages of a deterministic model.

For general performance models of parallel systems, the results imply that conclusions of such models that depend heavily on an exponential task assumption may not hold for many parallel programs. It is thus important to evaluate to what extent the conclusions of a model are dependent on such an assumption.

We also used the analytical model to prove that process execution times in the presence of random delays asymptotically approach a normal distribution, and we used direct measurements of the distribution of process execution times to show that, in practice, phases of many real programs exhibit a distribution that is very close to normal.

These results raise some issues that would be interesting to address in future. The preliminary results of Section 4.1 indicate that the deterministic approach to parallel program performance prediction could yield promising alternatives to previous stochastic models. Developing a generally applicable and efficient deterministic model based on these ideas, as well as a thorough evaluation of the model and comparisons with previous models is the subject of our current work. It would also be interesting to characterize the class of programs that do have significant variability in processing requirements, as we found for Locus Route. In particular, how common are such programs in practice? How can performance prediction models be extended to evaluate such programs? It may even be necessary to derive suitable performance metrics for such programs, since their execution time can vary significantly in different runs.

Acknowledgements

We are grateful to the Wisconsin Wind Tunnel group for access to and help with the Wind Tunnel software, which made our measurements on the CM-5 possible. We thank Eric Bach, Peter Haas, Rajesh Mansharamani and Sarita Adve for helpful comments and discussions about this work.

References

1. A. AGARWAL, R. SIMONI, M. HOROWITZ and J. HENNESSY, An Evaluation of Directory Schemes for Cache Coherence, *Proc. 15th Annual International Symposium on Computer Architecture*, Honolulu, Hawaii, June 1988, 280-289.
2. F. ALLEN, M. BURKE, P. CHARLES, R. CYTRON and J. FERRANTE, An overview of the ptran analysis system for multiprocessing, *Journal of Parallel and Distributed Computing 5*, 5 (October 1988), 617-640.
3. F. BACCELLI and Z. LIU, On the Execution of Parallel Programs on Multiprocessor Systems - A Queueing Theory Approach, *Journal of the ACM 37*, 2 (April 1990), 373-414.
4. E. D. BROOKS III, The indirect k-ary n-cube network for a vector processing environment, *Parallel Computing 6*(1988), 339-348.
5. C. S. CHANG and R. NELSON, Bounds on the Speedup and Efficiency of Partial Synchronization in Parallel Processing Systems, Research Report RC 16474, I.B.M., 1991.
6. M. DUBOIS and F. A. BRIGGS, Performance of Synchronized Iterative Processes in Multiprocessor Systems, *IEEE Trans. on Software Engineering SE-8*, 4 (July 1982), 419-431.
7. Y. O. FUENTES and S. KIM, Foundations of Parallel Computational Microhydrodynamics : Communication Scheduling Strategies, Research Report, Rheology Research Center, University of Wisconsin-Madison, May 1990.
8. A. GREENBAUM, Synchronization Costs on Multiprocessors, *Parallel Computing 10*(1989), 3-14.
9. A. KAPELNIKOV, R. R. MUNTZ and M. D. ERCEGOVAC, A Modeling Methodology for the Analysis of Concurrent Systems and Computations, *Journal of Parallel and Distributed Computing 6*(1989), 568-597.
10. C. P. KRUSKAL and A. WEISS, Allocating Independent Subtasks on Parallel Processors, *IEEE Trans. on Software Engineering SE-11*, 10 (October 1985), 1001-1016.
11. S. T. LEUTENEGGER and M. K. VERNON, The Performance of Multiprogrammed Multiprocessor Scheduling Policies, *Proc. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems 18*, 1 (May 1990), 226-236.
12. S. T. LEUTENEGGER and R. D. NELSON, Analysis of Spatial and Temporal Scheduling Policies for Semi-Static and Dynamic Multiprocessor Environments, *IBM Research Report*, August 1991.
13. G. LEWANDOWSKI, A. CONDON and E. BACH, Realistic Analysis of Parallel Dynamic Programming Algorithms, Computer Sciences Technical Report #1116, University of Wisconsin-Madison, October 1992.
14. S. MADALA and J. B. SINCLAIR, Performance of Synchronous Parallel Algorithms with Regular Structures, *IEEE Trans. on Parallel and Distributed Systems 2*, 1 (January 1991), 105-116.
15. V. W. MAK and S. F. LUNDSTROM, Predicting Performance of Parallel Computations, *IEEE Trans. on Parallel and Distributed Systems 1*, 3 (July 1990), 257-270.
16. J. MOHAN, *Performance of Parallel Programs: Model and Analyses*, Ph.D. Thesis, Carnegie Mellon University, July 1984.
17. R. NELSON, D. TOWSLEY and A. N. TANTAWI, Performance Analysis of Parallel Processing Systems, *IEEE Trans. on Software Engineering 14*, 4 (April 1988), 532-540.
18. R. NELSON, A. N. TANTAWI and D. TOWSLEY, The Order Statistics of the Sojourn Times of Customers that Form a Single Batch in the M^X/M/c Queue, Research Report RC 15141, I.B.M., 1990.
19. R. NELSON, A Performance Evaluation of a General Parallel Processing Model, *1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems 18*, 1 (May 1990), 14-26.
20. S. K. REINHARDT, M. D. HILL, J. R. LARUS, A. R. LEBECK, J. C. LEWIS and D. A. WOOD, The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers, *In these proceedings*, May 1993.
21. V. SARKAR, Determining Average Program Execution Times and their Variance, *Proc. 1989 SIGPLAN Notices Conference on Programming Language Design and Implementation*, 1989, 298-312.
22. S. SETIA and S. K. TRIPATHI, An Analysis of Several Processor Partitioning Policies for Parallel Computers, University of Maryland CS-Tech. Rep.-2684, May 1991.

23. J. P. SINGH, W. WEBER and A. GUPTA, SPLASH: Stanford Parallel Applications for Shared-Memory, *Computer Architecture News 20*, 1 (March 1992), 5-44.

24. R. E. TARJAN and U. VISHKIN, An Efficient Parallel Biconnectivity Algorithm, *SIAM Journal of Computing 14*, 4 (1985), 862-874.

25. A. THOMASIAN and P. F. BAY, Analytic Queueing Network Models for Parallel Processing of Task Systems, *IEEE Trans. on Computers C-35*, 12 (December 1986), 1045-1054.

26. K. S. TRIVEDI, *Probability and Statistics with Reliability, Queueing and Computer Science Applications*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

27. R. W. WOLFF, *Stochastic Modeling and the Theory of Queues*, Prentice-Hall, Englewood Cliffs, NJ, 1989.

28. J. ZAHORJAN and C. MCCANN, Processor Scheduling in Shared-Memory Multiprocessors, *Proc. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems 18*, 1 (May 1990), 214-225.

Appendix A. Proof that F_T is Asymptotically Normal

Claim. If σ_P^2 and σ_C^2 are finite, then $(T(D)-\mu(D))/\sigma(D) \Longrightarrow$ Normal(0,1) as $D \to \infty$, where $\mu(D)$ and $\sigma(D)$ are as given in (7).

Proof. The proof is essentially a direct application of the Central Limit Theorem for cumulative reward processes. For a renewal process generated by the i.i.d. sequence $\{X_i\}$ (with $E[X_1] = \mu$), let $\{W_i\}$ be a sequence of i.i.d. random variables (rewards), where W_i is independent of $\{X_j: j \neq i\}$. Define the cumulative reward $C(t)$ to be $C(t) \equiv \sum_{i=1}^{i=R(t)} W_i$, where $R(t)$ is the number of renewals up to time t (just as was defined in (4), Section 2.1). Then, if $E[X_j^2]$ and $E[W_j^2]$ are finite,

$$\frac{C(t) - \frac{1}{\mu} t\, E[W_1]}{\left[(t/\mu)\, \mathrm{Var}(W_1 - E[W_1]X_1/\mu)\right]^{1/2}} \Longrightarrow \text{Normal}(0,1) \text{ as } t \to \infty.$$

(A slightly more general case is proved in [27, p. 124].) In our model, let $X_i = P_i$, $W_i = C_i$. Then $\mu = \mu_P$, $E[W_1] = \mu_C$, and $C(D) = \sum_{i=1}^{R(D)} C_i$. Then, under the hypotheses of the claim, the above theorem directly applies, showing that $C(D)$ converges to a Normal with mean $D\mu_C/\mu_P$, and variance given by

$$\begin{aligned} \mathrm{Var}(C(D)) &= \frac{D}{\mu_P} \mathrm{Var}(C_1 - \frac{1}{\mu_P} E[C_1]P_1) \\ &= D\frac{\sigma_C^2}{\mu_P} + \frac{D\mu_C{}^2}{\mu_P{}^3} \sigma_P^2 \end{aligned}$$

But, $T = D + C(D)$, since $C(D)$ is just the total of the communication delays in the first $R(D)$ cycles. Thus T also converges to a Normal with mean $\mu_T = D + D\mu_C/\mu_P$ and variance σ_T^2 the same as the variance of $C(D)$ calculated above. **QED.**